APPENDIX A

1. Android Platform and Smartphone Experiments

I installed the Android SDK, the Eclipse desktop development environment, and the Android ADT plugin for Android development on my laptop. I also installed Android source code, software packages associated with the Android build environment, and built versions of the Android system on my laptop, and ran them on the Android emulator. Also, I used these tools to connect my laptop to the phone and examine the files that came installed on Android devices, including Nexus S (running Android Version 2.3.1), Nexus One (running Android Version 2.2), Samsung Captivate Galaxy S (running Android Version 2.2), HTC Droid Incredible 2 (running Android Version 2.2.1), LG Optimus S (running Android Version 2.2.1), and the Motorola Atrix (running Android Version 2.2.2).

Note that all files referenced in my analysis below are contained in Exhibit App-1, which is produced along with my report. These are locatable by the filenames or folder paths I identify below.

A. Laptop computer

I installed the software on a Dell Inspiron laptop purchased from Best Buy on 6/29/11. I opened the sealed package containing the laptop, and installed and configured software as described below.

B. Install Linux operating system

I installed Ubuntu Linux version 10.10 by using an install CD burned from an .iso image downloaded from the Ubuntu web site. In the installation process, I selected the *download updates* and *erase entire disk* options so that the installation process produced a single-boot laptop running Ubuntu Linux version 10.10. I configured this with a user name: *jcm* (my initials) and password *froyo* since my plan was to install and experiment with the Froyo version of Android.

C. Initialize the Android build environment

Overview information about installing the Android build environment is available at http://android.git.kernel.org/?p=platform/build.git;a=blob_plain;f=core/build-system.html. I followed the instructions for setting up the build environment at http://source.android.com/source/initializing.html, specifically, the instructions under the heading "Setting up a Linux build environment". These instructions call for installation of the Java Development Kit (JDK) and a number of standard Linux packages. Since Java 5 is required for Froyo, I downloaded Java 5. Since Ubuntu may include Java 6, I set my environment variables to match those for Java 5, so that Java 5 would be used in any source code programming or rebuild of the Android platform.

I installed several utilities, including the git version control system, by entering the following installation command specified in the web page above:

```
sudo apt-get install git-core gnupg flex bison gperf build-essential \
  zip curl zlib1g-dev libc6-dev lib32ncurses5-dev ia32-libs \
  x11proto-core-dev libx11-dev lib32readline5-dev lib32z-dev \
  libgl1-mesa-dev g++-multilib mingw32 tofrodos
```

To summarize, I used the commands in "get-gmail-apk-files-to-dexdump.txt" to create a new directory for each phone, *e.g.*, "htc-droid-gmail-apk". Then, I used the *find* command to find and copy files with names containing "Gmail" to the corresponding phone's gmail-apk directory. Next, I renamed each Gmail.apk to Gmail.zip (so that "unzip" works properly) and used "unzip" to extract a classes.dex file and other files from each Gmail.zip file. Finally, I used dexdump to create a dexdump file from the classes.dex file.

The commands saved in "string-at-other-phones-commands.txt" recorded the process for finding lines of source code that contain both "string@" and "value" (as a sample use of string@). This search found the .dex file constant pool indices, such as string@4ada, which is associated with the source code string "value". I selected "value" as an example because the "value" occurred in many classes and methods in Gmail.odex files. Because "value" is always associated with the same constant pool index in other Gmail .apk files, this constant is shared across many classes and methods.

The commands in "string-at-other-phones-commands.txt" repeat a sequence of steps for each phone. The last command in each of these sequences counts the occurrences of fill-array-data in the Gmail dexdump for that phone (i.e., the dexdump of classes.dex obtained by unzipping the Gmail.apk file).

M. Gmail source code examination (Google Highly Confidential—Attorneys' Eyes Only)

I examined the source code for Android's Gmail application, which was provided to me on paper with Bates numbers ranging from GOOGLE-SC-00000214 to GOOGLE-SC-00000497. I looked for static initializations of arrays of primitive types and occurrences of selected strings. I found several examples of static initializations of arrays of primitive types and compared them to uses of fill-array-data in dexdumps of the Gmail application. I also examined the Gmail application source code for examples of "Gmail" strings and found examples from different classes. I then looked at the corresponding occurrences of these strings in the dexdump file and determined that the same constant pool index is used for all of these occurrences.

2. Implications for Infringement Analysis

A. '104 patent

Log messages from modified build on emulator. I inserted calls into the logging functions in selected portions of the Android source code as described above. I *grep*-ed the file "adb-logcat-modified-build-1", which contains the log messages that the emulator generates as it starts up. While the output is lengthy, the log includes the following output lines, which show that log messages I inserted to test the execution of methods relevant to the '104 patent:

```
I/dalvikvm(  65): JCM --- dvmResolveClass in Resolve.c
I/dalvikvm(  33): JCM --- dvmResolveMethod in Resolve.c
I/dalvikvm(  33): JCM --- dvmResolveString in Resolve.c
I/dalvikvm(  61): JCM --- dvmOptResolveClass in DexOptimize.c
I/dalvikvm(  61): JCM --- dvmOptResolveMethod in DexOptimize.c
I/dalvikvm(  61): JCM --- dvmOptResolveInterfaceMethod in DexOptimize.c
I/dalvikvm(  61): JCM --- dvmOptResolveStaticField in DexOptimize.c
I/dalvikvm(  87): JCM --- optimizeLoadedClasses in DexOptimize.c
```

I/dalvikvm(61): JCM --- optimizeMethod in DexOptimize.c

Files on Android devices. All six phones contain a file called "/system/lib/libdvm.so". In all of the files with this name, the following function names were found by using *readelf*: dvmResolveClass, dvmResolveMethod, dvmResolveString, dvmOptResolveClass, dvmOptResolveMethod, dvmOptResolveInterfaceMethod, and dvmOptResolveStaticField. In the Android Froyo source code, the first three of these functions are defined in the file Resolve.c and the other four are defined in the file DexOptimize.c (and only in these source code files). In Gingerbread, these functions may be present in different source code files because the Android source code was partially restructured between Froyo and Gingerbread versions. This provides evidence that the object code compiled from these source files is present on these Android devices.

B. '205 patent

Log messages from modified build on emulator. I inserted calls into logging functions in selected portions of the Android source code as described above. I *grep*-ed the file "adb-logcat-modified-build-1", which contains the log messages that the emulator generates as it starts up. While the output is lengthy, the log includes the following output lines, which show the log messages I inserted to test the execution of methods relevant to the '205 patent:

I/dalvikvm(87): JCM --- optimizeLoadedClasses in DexOptimize.c
I/dalvikvm(61): JCM --- optimizeClass in DexOptimize.c
I/dalvikvm(61): JCM --- optimizeMethod in DexOptimize.c
I/dalvikvm(61): JCM --- rewriteExecuteInlineRange in DexOptimize.c
I/dalvikvm(87): JCM --- createInlineSubsTable in DexOptimize.c

Log messages operating Android devices. I examined the boot log from each of the phones to see if JIT-related log messages occurred. The "JIT started" log message appears on logs from Nexus S, Nexus One, Samsung Galaxy S, and HTC Droid Incredible 2. The "Jit: resizing JitTable" log message appears on logs from all of the tested devices (Nexus S, Nexus One, LG Optimus S, Motorola Atrix, Samsung Galaxy S, and HTC Droid Incredible 2). Searching the Froyo source code (by using the command "*grep*") I found that the log command producing "JIT started…" occurs in the file dalvik/vm/compiler/Compiler.c and no other source code file. A search of the source code shows that the log command producing "resizing JitTable …." occurs in the file dalvik/vm/interp/Jit.c and no other source code file. This provides evidence that the object code compiled from dalvik/vm/compiler/Compiler.c is present and executed on most of the devices and that the file dalvik/vm/interp/Jit.c is present and executed on all of the Android devices.

To give further examples of the log files and their contents, a "*grep -i*" for JIT in the script file "htc-droid-2-boot-logcat" showed the following logged activity:

D/dalvikvm(1457): JIT started for system_server
I/dalvikvm(1457): Jit: resizing JitTable from 4096 to 8192
I/dalvikvm(1577): Total arena pages for JIT: 11

following methods are executed: method preloadClasses from the file ZygoteInit.java, Dalvik_dalvik_system_Zygote_forkAndSpecialize fromthe file dalvik_system_Zygote.c, dvmLinkClass from the file Class.c, and method runOnce from the file ZygoteConnection.java.

Log messages on Android devices. I examined the boot log from each of the phones to see if zygote-related log messages occurred. The "…com.android.internal.os.ZygoteInit" log message appears on logs from the Nexus S, Motorola Atrix, and HTC Droid Incredible 2. The "Zygote… Preloading classes" log message appears on logs from all of the tested devices (Nexus S, Nexus One, LG Optimus S, Motorola Atrix, Samsung Galaxy S, and HTC Droid Incredible 2).

Files on Android devices. All six phones contain a file called /system/lib/libdvm.so. In all of the files with this name, the following names were found using *readelf*: dvmLinkClass and dvm_dalvik_system_Zygote. In the Froyo source code, the function dvmLinkClass is defined in the Class.c and the object dvm_dalvik_system_Zygote is defined in the file dalvik_system_Zygote.c (and only in these source code files). The DalvikNativeMethod object dvm_dalvik_system_Zygote associates forkAndSpecialize with the native static function Dalvik_dalvik_system_Zygote_forkAndSpecialize, as indicated in the C source code file. In Gingerbread, these functions may be in different source code files because the code may have been partially restructured. This provides evidence that object code compiled from these source files is present on these Android devices.

All six phones contain a file called /system/framework/framework.odex. In all of the files with this name, the following names were found using dexdump: preloadClasses and runOnce. In the Android Froyo source code, the first is defined in ZygoteInit.java and the second in ZygoteConnection.java. In Gingerbread, these functions may be in different source code files because the Android source code was partially restructured between Froyo and Gingerbread VERSIONS. This provides evidence that bytecode compiled from these source files is present on these Android devices.

F. ’447 and ’476 patents

Log messages from modified build on emulator. I inserted calls into logging functions in selected portions of the Android source code as described above. I *grep*-ed the file ”adb-logcat-cts-1“ which contains the log messages that the emulator generates as I ran the CTS test described above. The output shows that the checkPermission method defined in the file AccessControlContext.java and the checkPermission method defined in the file AccessController.java are called in this run of the CTS test.

Files on Android devices. All six phones contain a file called /system/lib/libdvm.so. In all of the files with this name, the following names were found using *readelf*: dvmIsPrivilegedMethod, dvm_java_security_AccessController, and dvm_java_lang_VMClassLoader. In the Froyo source code, the function dvmIsPrivilegedMethod is defined in the file InternalNative.c (and only in this source code file). The object dvm_java_security_AccessController is defined in the file java_security_AccessController.c, and the object dvm_java_lang_VMClassLoader is defined in the file java_lang_VMClassLoader.c. The DalvikNativeMethod object dvm_java_security_AccessController associates getStackDomains with the native static function Dalvik_java_security_AccessController_getStackDomains, as indicated in the C source code file. The

DalvikNativeMethod object dvm_java_lang_VMClassLoader associates defineClass with the native static function Dalvik_java_lang_VMClassLoader_defineClass, as indicated in the C source code file. The DalvikNativeMethod object dvm_java_lang_VMClassLoader associates defineClass (with a different method signature) with the native static function Dalvik_java_lang_VMClassLoader_defineClass2, as indicated in the C source code file. In Gingerbread, these functions may be in different source code files because the Android source code was partially restructured between Froyo and Gingerbread versions. This provides evidence that object code compiled from these source files is present on these Android devices.

All six phones contain a file called /system/framework/framework.odex. In all of the files with this name, the following names were found using *dexdump*: checkPermission and implies. In the Froyo source code, the checkPermission is defined in the files AccessController.java, AccessControlContext.java and SecurityManager.java (and only these Java source code files); implies is defined in the file ProtectionDomain.java (and only this Java source code file). In Gingerbread, these functions may be in different source code files because the Android source code was partially restructured between Froyo and Gingerbread versions.